Virtualization Technologies

CPSC 454 Cloud Computing & Security

Portions of this PPT draw from PPT authored by Professor Dijiang Huang at Arizona State University

Outline

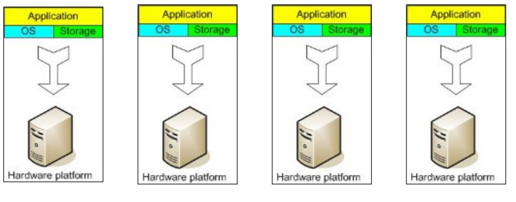
- Virtualization Concepts and Classifications
- I/O Virtualization
- VMM (Virtual Machine Monitor)

Virtualization Concepts and Classification

Virtualization: What is it, really?

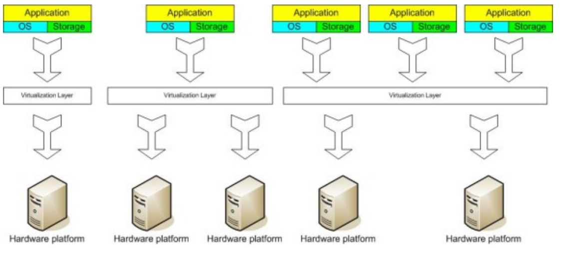
- Real vs. Virtual
  - Similar essence, effect
  - *"Formally"* different
- A framework that combines or divides [computing] resources to present a *transparent* view of one or more environments
  - Hardware/software partitioning (or aggregation)
  - Partial or complete machine simulation
  - Emulation (again, can be partial or complete)
  - Time-sharing (in fact, sharing in general)
  - In general, can be M-to-N mapping (M "real" resources, N "virtual" resources)
    - Examples: VM (M-N), Grid Computing (M-1) , Multitasking (1-N)

The Traditional Server Concept



Web Server	App Server	DB Server	EMail
Windows	Linux	Linux	Windows
IIS	Glassfish	MySQL	Exchange

The Virtual Server Concept



Virtual Machine Monitor (VMM) layer between *Guest OS* and hardware

Virtualization Products

Implementation	Virtualization Type	Installation Type	License
Bochs	Emulation	Hosted	LGPL
QEMU	Emulation	Hosted	LGPL/GPL
VMware	Full Virtualization & Paravirtualization	Hosted and bare-metal	Proprietary
User Mode Linux(UML)	Paravirtualization	Hosted	GPL
Lguest	Paravirtualization	Bare-metal	
OpenVZ	OS level	Bare-metal	GPL
Linux VServer	OS level	Bare-metal	GPL
Xen	Paravirtualization or Full when using hardware extensions	Bare-metal	GPL
Parallels	Full Virtualization	Hosted	Proprietary
Microsoft	Full Virtualization	Hosted	Proprietary
z/VM	Full Virtualization	Hosted and bare-metal	Proprietary
KVM	Full Virtualization	Bare-metal	GPL
Solaris Containers	OS level	Hosted	CDDL
BSD Jails	OS level	Hosted	BSD
Mono	Application Layer	Application Layer	Compiler and tools GPL, Runtime libraries LGPL, Class libraries MIT X11
Java Virtual Machine	Application Layer	Application Layer	GPL

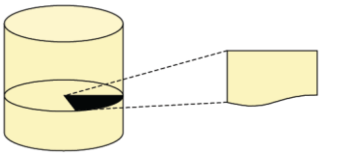
Virtualization: Why?

- Server consolidation
- Application Consolidation
- Sandboxing
- Multiple execution environments
- Virtual hardware
- Debugging
- Software migration (Mobility)
- Appliance (software)
- Testing/Quality Assurance

Virtualization

- A layer mapping its visible interface and resources onto the interface and resources of the underlying layer or system on which it is implemented
- Purposes
  - Abstraction – to simplify the use of the underlying resource (e.g., by removing details of the resource's structure)
  - Replication – to create multiple instances of the resource (e.g., to simplify management or allocation)
  - Isolation – to separate the uses which clients make of the underlying resources (e.g., to improve security)

Abstraction



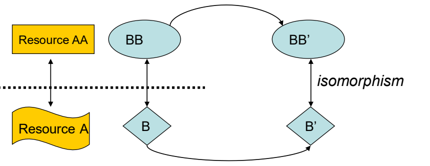
- Abstraction is about hiding details
  - a file on a hard disk
    - mapped to a collection of sectors and tracks on the disk
    - we don't directly address disk layout when accessing the file
  - a level of abstraction provides a simplified interface to underlying resources

Abstraction

- **Why Abstraction?**
  - it is a key to managing complexity in computer systems
    - different abstraction levels: hierarchy + interfaces
  - interfaces simplify your life!
    - no need to deal with too many details
    - create a certain level of vendor independence
- **Abstraction Limitations**
  - diversity reduces interoperability
  - instruction sets, operating systems, programming languages
  - operating system designs reduce flexibility
    - operating systems introduce abstractions for memory, I/O, ...
    - typical approach: the OS manages resources directly
    - implicit assumption: all resources under a single regime
    - limited flexibility wrt. applications, security, failure isolation

Abstraction, Virtualization of Computer System

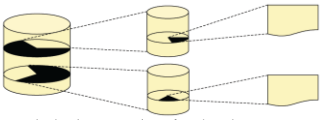
- Virtualization
  - Similar to Abstraction but it does not always hide low layer's details
  - Real system is transformed so that it appears to be different



  - Virtualization can be applied not only to subsystem, but to an *Entire Machine* → ***Virtual Machine***

Virtualization

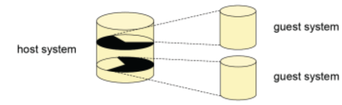
- Virtualization is about creating illusions



  - two files on separate hard disks, each of which is a partition on an actual hard disk
  - virtualization:
    - o a virtualized system's interface and resources are mapped onto interface and resources of another ("real") system
    - o virtualization provides a different interface and/or resources at the same level of abstraction

13

Virtualization

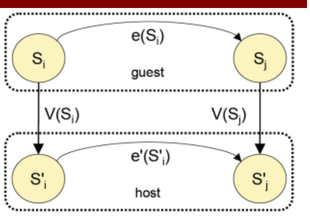


- two types of systems involved
  - the virtualized system is called guest system
  - the "real" system is called host system
- multiple layers of virtualization
  - virtualized host system is also an option

14

Virtualization

- the host is transformed
  - appears to be a (set of) different *virtualized* system(s)
  - introduction of an isomorphism mapping guest to host system



Si, Sj: states in a system
e: operation sequence modifying Si to Sj
V: function mapping guest states to host states
S'i, S'j: host states
e': operation sequence corresponding to e

- virtualization vs. abstraction
  - the same isomorphism can also be used to depict abstraction
  - important difference: virtualization does not necessarily hide details

15

What Can be Virtualized

- Computers
- Storage
- Network
- Services
- Security
- … almost everything
- What we see the mostly is to virtualize a computer.

Virtual Machine

- **"Virtual"**
  - what we know now
    - o virtualization is about creating illusions
    - o map interfaces and resources to other interfaces and resources
- **"Machine"?**
  - what we need to know: notion of "machine"
    - o what is virtualized to build a virtual machine?
    - o what interfaces and resources are there that can be subject to
- **virtualization?**
  - roadmap
    - o look at computer system architecture
    - o identify (types of) interfaces
    - o identify and characterize virtualization approaches

17

Virtualization is "HOT"

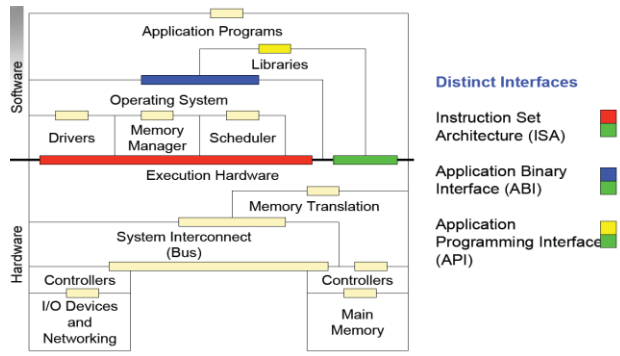
- Microsoft acquires Connectix Corp.
- EMC acquires VMware
- Veritas acquires Ejascent
- IBM, already a pioneer
- Sun working hard on it
- HP picking up

➔Virtualization is HOT!!!

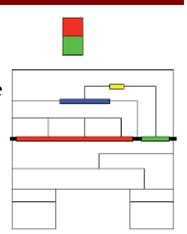
First Virtualization Classification

- ISA based virtualization
- ABI based virtualization
- API based virtualization

Computer Architecture

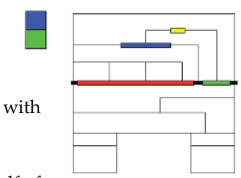


Instruction Set Architecture



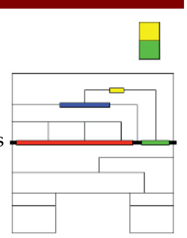
- division between hardware and software
- sub-parts
  - user ISA parts of the ISA visible to applications
  - system ISA parts of the ISA visible to supervisor software
  - system ISA can also employ user ISA components
- software compatibility
  - software built to a given ISA can run on any hardware that supports that ISA

Application Binary Interface (ABI)



- provides programs with access to hardware resources and services
- major components
  - set of all user instructions
  - system calls : indirect interaction with hardware resources
- system calls
  - OS operations performed on behalf of user programs
  - often includes security checks (wrt. access privileges)
- support for portability
  - binaries compiled to a specific ABI can run unchanged on a system with the same ISA and OS

Application Programming Interface (API)



- abstracts from the details of service implementations
- usually defined with respect to a high-level language (HLL)
  - standard library to invoke OS services
  - typically defined at source code level
- support for portability
  - software using a given API can be ported to other platforms by recompilation

"Machine": Matter of Perspective

Process Perspective

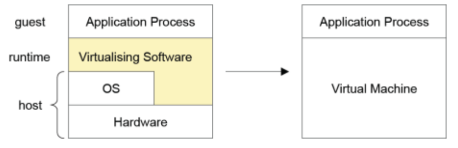
- machine = combination of OS and user-level
  - logical memory address space assigned to the process
  - user-level registers and instructions for program execution
  - I/O part of machine visible only through OS; interaction via OS calls
- ABI provides process/machine interface

Operating System Perspective

- machine is implemented by underlying alone
  - system: full execution environment for multiple users
  - processes share file system and other resources
  - OS is part of the system
- ISA provides system/machine interface

Process Virtual Machines

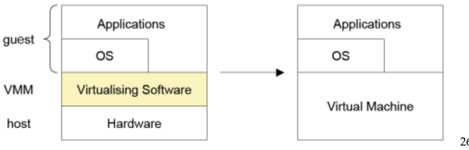
- capable of supporting an individual process
- virtualizing software
  - placed at ABI, on top of OS + hardware
  - emulates both user-level instructions and OS calls



25

System Virtual Machines

- provide a complete system environment
  - can support a "guest OS" with (probably) many user processes
- virtualizing software ("virtual machine monitor", VMM)
  - placed between underlying hardware and conventional software
  - ISA translation
  - alternative approach: hosted VM (virtualizing software built *on top of* an OS)



26

Second Virtualization Classification

- Instruction Set Architecture
  - Emulate the ISA in software
    - o Interprets, translates to host ISA (if required)
    - o Device abstractions implemented in software
    - o Inefficient
  - Optimizations: Caching, Code reorganization.
  - Applications: Debugging, Teaching, multiple OS
- Hardware Abstraction Layer (HAL)
  - Between "real machine" and "emulator" (maps to real hardware)
  - Handling non-virtualizable architectures
  - Applications: Fast and usable, virtual hardware, consolidation, migration

Second Virtualization Classification
cont'd

- Operating System Level
  - Virtualized SysCall Interface
  - May or may not provide all the device abstractions
  - Easy to manipulate (create, configure, destroy)
- Library (user-level API) Level
  - Presents a different subsystem API to application
  - Complex implementation, if kernel API is limited
  - User-level device drivers
- Application (Programming Language) Level
  - Virtual architecture (ISA, registers, memory, …)
  - Platform-independence (➔ highly portable)
  - Less control on the system (extremely high-level)

Overall Picture

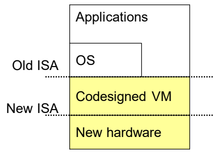
	ISA	HAL	OS	Library	PL
Performance	*	****	****	***	**
Flexibility	****	***	**	**	**
Ease of Impl	**	*	***	**	**
Degree of Isolation	***	****	**	**	***

(more stars are better)

Instruction Set Architecture Level Virtualization

- Technologies
  - Emulation: Translates guest ISA to native ISA
  - Emulates h/w specific IN/OUT instructions to mimic a device
  - Translation Cache: Optimizes emulation by making use of similar recent instructions
  - Code rearrangement
  - Speculative scheduling (alias hardware)
- Issues
  - Efficient Exception handling
  - Self-modifying code

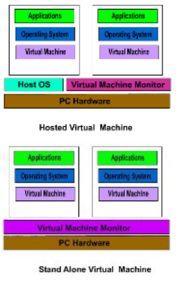
ISA Level Virtualization: Examples



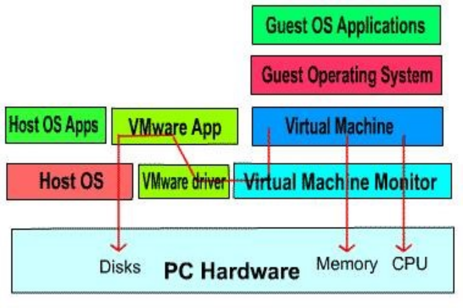
- Bochs: Open source x86 emulator
  - Emulates whole PC environment
    - x86 processor and most of the hardware (VGA, disk, keyboard, mouse, …)
    - Custom BIOS, emulation of power-up, reboot
    - Host ISAs: x86, PowerPC, Alpha, Sun, and MIPS
- Crusoe (Transmeta)
  - "Code morphing engine" – dynamic x86 emulator on VLIW processor
  - 16 MB "translation cache"
  - Shadow registers: Enables easy exception handling
- QEMU:
  - Full Implementation
    - Multiple target ISAs: x86, ARM, PowerPC, Sparc
    - Supports self-modifying code
    - Full-software and simulated (using mmap()) MMU
  - User-space only: Useful for Cross-compilation and cross-debugging

HAL Virtualization Techniques

- Standalone vs. Hosted
  - Drivers
  - Host and VMM worlds
  - I/O
- Protection Rings
  - Multilevel privilege domains
- Handling "silent" fails
  - Scan code and insert/replace artificial traps
  - Cache results to optimize



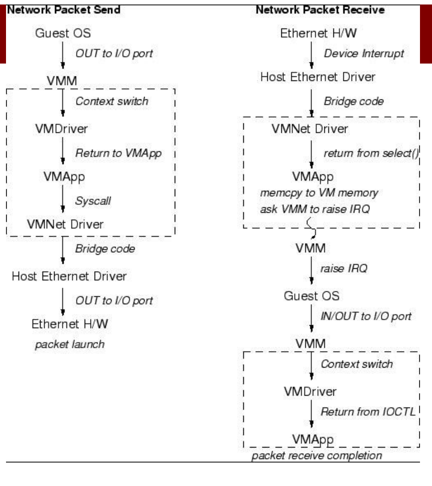
VMware Architecture



VMware Workstation Architecture

VMware: I/O Virtualization

- VMM does not have access to I/O
- I/O in "host world"
  - Low level I/O instructions (issued by guest OS) are merged to high-level I/O system calls
  - VM Application executes I/O SysCalls
- VM Driver works as the communication link between VMM and VM Application
- World switch needs to "save" and "restore" machine state
- Additional techniques to increase efficiency



Paravirtualization

- Traditional architectures do not scale
  - Interrupt handling
  - Memory management
  - World switching
- Virtualized architecture interface
  - Much simpler architectural interface
  - Virtual I/O and CPU instructions, registers, …
- Portability is lost

Paravirtualizing the Memory Management Unit (MMU)

- Guest OS allocate and manage own page-tables
  - Hypercalls to change PageTable base.
- Xen Hypervisor is responsible for trapping accesses to the virtual page table, validating updates and propagating changes.
- Xen must validate page table updates before use
  - Updates may be queued and batch processed
- Validation rules applied to each PTE
  - Guest may only map pages it owns
- XenoLinux implements a balloon driver
  - Adjust a domain's memory usage by passing memory pages back and forth between Xen and XenoLinux

37

Examples

- Denali
  - Simpler customized OS with no VM for network applications
- Xen
  - Simpler port to commercial OS
  - Exposes some "real" hardware, e.g. clock, physical memory address

OS Level Virtualization

- Containers (operating environments) on top of OS
  - Processes, File System, Network resource (IP address), Environment variables, System call interface
- Technologies
  - chroot(): File system virtualization on Unix
  - Name spaces: Each container is tagged and new entities (fork()) generated from a container remains inside
  - System call interposition: The only interface with user space, can modify parameters, return values (to expose a different environment)
  - Copy-on-write: Enables sharing of files
- Applications: Sandboxing, Fine grain access control (root in the container)

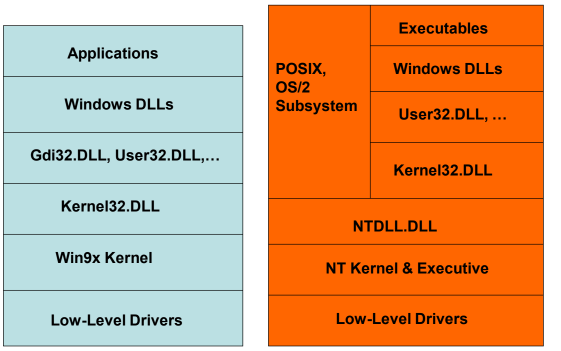
OS Level Virtualization: Examples

- Jail
  - FreeBSD based virtualization using "*chroot()*"
  - Scope is limited to the *jail*
  - Curtailed access to resources and operations
    - Signals, debugger, IP spoofing, system calls
  - A file-system sub-tree, one IP address, one "root"
- Ensim's "Virtual Private Server"
  - Supports virtual "boot", per-VM resource limits
  - Virtual /proc, IP address-space
- Linux "Virtual Environment" (VE)
  - Tagged VE (VE-id), policy support for the rights of "root"

Library Level Virtualization

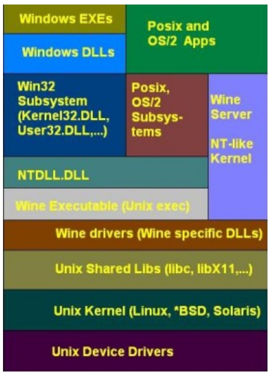
- Technologies
  - API interception through DLL hooking
  - Partial/complete implementation of APIs
  - Emulate low level kernel implementations in user-space
    - Useful when the host OS does not provide required support (e.g. Win32 threads vs. pthreads)
    - Mandatory drivers
- Examples
  - WINE: Win32 API implementation on Unix/X
  - POSIX, OS/2 subsystems on Windows
    - Supports Unix and OS/2 like API
  - LxRun: Linux API implementation on SCO UnixWare, Solaris
  - WABI: Sun's implementation similar to WINE (not extensive)

Windows Architecture



Wine Architecture

- Closely follows NT
  - Implements all the "core" DLLs (ntdll, user32, kernel32)
- Wine server provides the NT backbone
  - Message passing
  - Synchronization
  - Object handles
- Native DLL support for non-core libraries
- Hardware access through Unix device drivers



WINE Implementation

- Wine server
  - IPC through Unix sockets and shared message queues
  - Process/Thread management
  - Simulates Synchronization primitives
- Native vs. Built-in DLLs
  - DLLs are implemented as Unix shared libraries (built-in DLLs)
  - Supports non-core Windows DLLs (Native DLLs)
  - A fully implemented built-in DLL takes precedence over native DLLs
- Executable Load
  - DLL descriptors table maintain the list of loaded DLLs
  - Imports are resolved using DLL descriptor table or on-disk DLLs
- Processes/Threads
  - Windows processes are mapped to WINE/UNIX processes
  - Thread-related APIs implemented in user-space and using pthreads

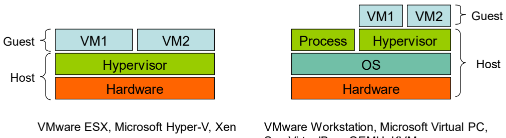
Application Level Virtualization

- Java Virtual Machine (JVM)
  - Executes Java byte code (virtual instructions)
  - Provides the implementation for the instruction set interpreter (or JIT compiler)
  - Provides code verification, SEH, garbage collection
  - Hardware access through underlying OS
- JVM Architecture
  - Stack-based architecture
  - No MMU
  - Virtual hardware: PC, register-set, heap, method (code) areas
  - Rich instruction set
    - o Direct object manipulation, type conversion, exception throws
- Provides a runtime environment through JRE
- Other Examples: .NET CLI, Parrot (PERL 6)

Type	Description	Advantages	Disadvantages
Emulation	The hypervisor presents a complete virtual machine (of a foreign computing architecture to the host) enabling foreign applications to run in the emulated environment.	Simulates hardware that is not physically available.	Low performance and low density
Full	The hypervisor provides a complete virtual machine (of the same computing architecture as the host) enabling unmodified guests to run isolation.	Flexibility-run different versions of different operating systems from multiple vendors.	Guest OS does not know that it is being virtualized. Can incur a sizable performance hit on commodity hardware, particularly for I/O intensive applications.
Para	The hypervisor provides a complete but specialized virtual machine (of the same computing architecture as the host) to each guest allowing modified guests to run in isolation.	Lightweight and fast, near native speeds: Demonstrated to operate in the 0.5%-3.0% overhead range. [http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf] Allows OS to cooperate with hypervisor – improves IO and resource scheduling. Allows virtualizing architecture that do not support full virtualization.	Requires porting quest OS to use hypercalls instead of sensitive instructions. The main limitataion of paravirtualization is the guest OS must be tailored specifically to run on top of the virtual machine monitor(VMM), the host program that support multiple, identical execution environments. This especially impacts legacy closed source operating systems that have not yet implemented paravirtualized extensions.
OS level	A single operating system is modified in such a way as to allow various user space server processes to be coalesced into functional unites, which are executed in isolation from one another while running on a single hardware platform.	Fast, lightweight virtualization layer. It has the best possible (that is, close to native) performance and density, and features dynamic resource and management.	In practice, strong isolation is difficult to implement. Requires the same OS and patch level on all virtualized machines (homogeneous computing infrastructure).
Library	Emulates operating systems or subsystems via a special software library. Does not provide the illusion of a stand-alone system with a full operating system.	Provides missing API for application developers.	Often performs more slowly than a native optimized port of the application.
Application	Applications run in a virtual execution environment that provides a standard API for cross-platform execution and manages the application's consumption of local resources.	Manages resources automatically, which eases programmer learning curve. Increase portability of applications.	Execution is slower than native code. Overhead of virtual machine incurred when compared to native code.

Third Classification: Two types of hypervisors

- Definitions
  - **Hypervisor** (or **VMM** – Virtual Machine Monitor) is a software layer that allows several **virtual machines** to **run** on a **physical machine**
  - The physical OS and hardware are called the **Host**
  - The virtual machine OS and applications are called the **Guest**



VMware ESX, Microsoft Hyper-V, Xen

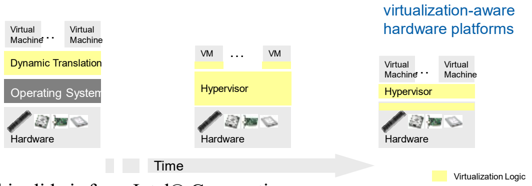
VMware Workstation, Microsoft Virtual PC, Sun VirtualBox, QEMU, KVM

Bare-metal or hosted?

- **Bare-metal**
  - Has complete **control over hardware**
  - Does not have to **"fight" an OS**
- **Hosted**
  - Avoid **code duplication**: need not code a **process scheduler**, **memory management** system – the **OS already does** that
  - Can run native **processes alongside** VMs
  - Familiar environment – **how much CPU** and **memory** does a VM take? Use **top**! How big is the **virtual disk**? **ls –l**
  - Easy management – stop a VM? Sure, just kill it!
- A combination
  - Mostly hosted, but some parts are inside the OS kernel for performance reasons
  - E.g., **KVM**

Evolution of Software solutions*

- 1st Generation: Full virtualization (Binary rewriting)
  - Software Based
  - VMware and Microsoft
- 2nd Generation: Paravirtualization
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- 3rd Generation: Silicon-based (Hardware-assisted) virtualization
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms



*This slide is from Intel® Corporation

I/O Virtualization

I/O Virtualization

- We saw **methods** to **virtualize** the **CPU**
- A computer is more than a CPU
- Also need I/O!

- Types of I/O:
  - Block (e.g., hard disk)
  - Network
  - Input (e.g., keyboard, mouse)
  - Sound
  - Video
- Most performance critical (for servers):
  - Network
  - Block

Side note – How does a NIC (network interface card) driver work?

- Transmit path:
  - OS prepares packet to transmit in a buffer in memory
  - Driver writes **start address** of buffer to **register X** of the NIC
  - Driver writes **length** of buffer to **register Y**
  - Driver writes '1' (**GO!**) into **register T**
  - NIC reads packet from memory addresses [X,X+Y) and sends it on the wire
  - NIC sends interrupt to host (**TX complete**, next packet please)
- Receive path:
  - Driver prepares buffer to receive packet into
  - Driver writes **start address** of buffer to **register X**
  - Driver writes **length** of buffer to **register Y**
  - Driver writes '1' (**READY-TO-RECEIVE**) into **register R**
  - When packet arrives, NIC copies it into memory at [X,X+Y)
  - NIC interrupts host (**RX**)
  - OS processes packet (e.g., wake the waiting process up)

I/O Virtualization? Emulate!

- Hypervisor implements **virtual NIC** (by the specification of a real NIC, e.g., Intel, Realtek, Broadcom)
- NIC **registers** (X, Y, Z, T, R, …) are just **variables** in hypervisor (host) **memory**
- If **guest writes** '1' to **register T**, **hypervisor reads** buffer from **memory [X,X+Y)** and passes it to **physical NIC** driver for transmission
- When physical NIC interrupts (**TX complete**), hypervisor **injects** TX complete interrupt into guest

- Similar for RX path

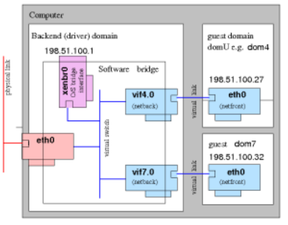
I/O Virtualization? Emulate!

- Pro:
  - **Unmodified guest** (guest already has drivers for Intel NICs…)

- Cons:
  - **Slow** – every access to every NIC register causes a **VM exit** (trap to hypervisor)
  - Hypervisor needs to **emulate complex hardware**

- Example hypervisors: QEMU, KVM, VMware (without VMware Tools)

I/O Virtualization? Paravirtualize!



- Add virtual NIC driver into guest (**frontend**)
- Implement the virtual NIC in the hypervisor (**backend**)
- Everything works just like in the emulation case…
- …except – **protocol** between frontend and backend
- Protocol in emulation case:
  - Guest writes registers X, Y, waits at least 3 nano-sec and writes to register T
  - Hypervisor **infers** guest wants to transmit packet
- Paravirtual protocol:
  - Guest does a hypercall, passes it start address and length as arguments
  - Hypervisor **knows** what it should do
- Paravirtual protocol can be **high-level**, e.g., ring of buffers to transmit (so NIC doesn't stay idle after one transmission), and **independent of particular NIC** registers

55

I/O Virtualization? Paravirtualize!

- Pro:
  - **Fast** – no need to emulate physical device
- Con:
  - Requires **guest driver**
- Example hypervisors: QEMU, KVM, VMware (with VMware Tools), Xen
- How is paravirtual I/O different from paravirtual guest?
  - Paravirtual guest requires to modify **whole OS**
    - Try doing it on Windows (without source code), or even Linux (lots of changes)
  - Paravirtual I/O requires the addition of a **single driver** to a guest
    - Easy to do on both Windows and Linux guests

56

I/O Virtualization? Direct access / direct assignment!

- "Pull" NIC out of the host, and "plug" it into the guest
- Guest is allowed to access NIC registers **directly**, no hypervisor intervention
- Host can't access NIC anymore

57

I/O Virtualization? Direct access / direct assignment!

- Pro:
  - As **fast** as possible!
- Cons:
  - Need NIC per guest
  - Plus one for host
  - Can't do "cool stuff"
    - Encapsulate guest packets, monitor, modify them at the hypervisor level
- Example hypervisors: KVM, Xen, VMware

58

I/O Virtualization? Emerging standard – SR-IOV!

- Single root I/O virtualization
- Contains a **physical function** controlled by the host, used to create **virtual functions**
- Each virtual function is assigned to a guest (like in **direct assignment**)
- Each **guest thinks** it has **full control** of NIC, accesses registers directly
- NIC does multiplexing/demultiplexing of traffic

59

I/O Virtualization? Emerging standard – SR-IOV!

- Pros:
  - As **fast** as possible!
  - Need only one NIC (as opposed to direct assignment)
- Cons:
  - Emerging standard
    - Few hypervisors fully support it
    - Expensive!
    - Requires new hardware
  - Can't do "cool stuff"
- Example hypervisors: KVM, Xen, VMware

60

Industry trends on I/O virtualization

- SR-IOV is the fastest, however it is the most expensive solution
- Paravirtual I/O is cheap, but much worse performance
- Companies (Red Hat, IBM, …) are looking at paravirtual I/O, trying to optimize it

61

Virtual Machine Monitor (VMM)

Concepts

References and Sources

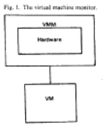
- James Smith, Ravi Nair, "The Architectures of Virtual Machines," IEEE Computer, May 2005, pp. 32-38.
- Mendel Rosenblum, Tal Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," IEEE Computer, May 2005, pp. 39-47.
- L.H. Seawright, R.A. MacKinnon, "VM/370 – a study of multiplicity and usefulness," IBM Systems Journal, vol. 18, no. 1, 1979, pp. 4-17.
- S.T. King, G.W. Dunlap, P.M. Chen, "Operating System Support for Virtual Machines," Proceedings of the 2003 USENIX Technical Conference, June 9-14, 2003, San Antonio TX, pp. 71-84.
- A. Whitaker, R.S. Cox, M. Shaw, S.D. Gribble, "Rethinking the Design of Virtual Machine Monitors," IEEE Computer, May 2005, pp. 57-62.
- G.J. Popek, and R.P. Goldberg, "Formal requirements for virtualizable third generation architectures," CACM, vol. 17 no. 7, 1974, pp. 412-421.

Definitions

- Virtual Machine Monitor (VMM)
  - A virtualization system that partitions a single physical "machine" into multiple virtual machines.
  - Terminology
    - Host – the machine and/or software on which the VMM is implemented
    - Guest – the OS which executes under the control of the VMM

Origins - Principles

Fig. 1. The virtual machine monitor.



"an *efficient, isolated duplicate* of the real machine"

- Efficiency
  - Innocuous instructions should execute directly on the hardware
- Resource control
  - Executed programs may not affect the system resources
- Equivalence
  - The behavior of a program executing under the VMM should be the same as if the program were executed directly on the hardware (except possibly for timing and resource availability)

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Communications of the ACM, vol 17, no 7, 1974, pp.412-421

Origins - Principles

Instruction types

- Privileged
  an instruction traps in unprivileged (user) mode but not in privileged (supervisor) mode.
- Sensitive
  - Control sensitive –
    attempts to change the memory allocation or privilege mode
  - Behavior sensitive
    - Location sensitive – execution behavior depends on location in memory
    - Mode sensitive – execution behavior depends on the privilege mode
- Innocuous – an instruction that is not sensitive

Theorem

For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

Signficance

The IA-32/x86 architecture is not virtualizable.

Origins - Technology

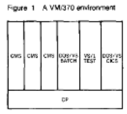
VM/370—a study of multiplicity and usefulness

by L. H. Seawright and R. A. MacKinnon

The productivity of data processing professionals and other professionals can be enhanced through the use of interactive and time-sharing systems. Similarly, system programmers can benefit from the use of system testing tools. A systems solution to both areas can be the virtual machine concept, which provides multiple software replicas of real computing systems on one real processor. Each virtual machine has a full complement of input/output devices and provides functions similar to those of a real machine. One system that implements virtual machines is IBM's Virtual Machine Facility/370 (VM/370).[1]

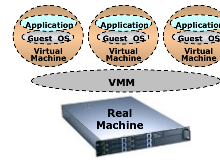
IBM Systems Journal, vol. 18, no. 1, 1979, pp. 4-17.

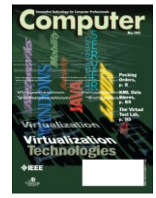
Figure 1 A VM/370 environment



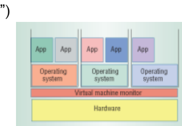
- Concurrent execution of multiple production operating systems
- Testing and development of experimental systems
- Adoption of new systems with continued use of legacy systems
- Ability to accommodate applications requiring special-purpose OS
- Introduced notions of "handshake" and "virtual-equals-real mode" to allow sharing of resource control information with CP
- Leveraged ability to co-design hardware, VMM, and guestOS

VMMs Rediscovered



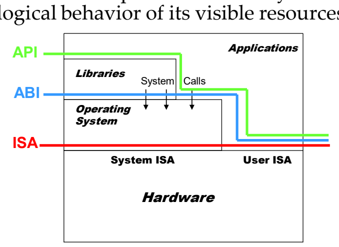


- Server/workload consolidation (reduces "server sprawl")
- Compatible with evolving multi-core architectures
- Simplifies software distributions for complex environments
- "Whole system" (workload) migration
- Improved data-center management and efficiency
- Additional services (workload isolation) added "underneath" the OS
  - security (intrusion detection, sandboxing,…)
  - fault-tolerance (checkpointing, roll-back/recovery)



Architecture & Interfaces

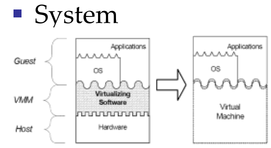
- Architecture: formal specification of a system's interface and the logical behavior of its visible resources.



- **API** – application binary interface
- **ABI** – application binary interface
- **ISA** – instruction set architecture

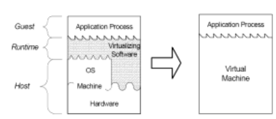
VMM Types

- System



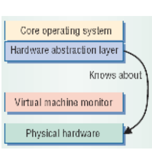
- Provides ABI interface
- Efficient execution
- Can add OS-independent services (e.g., migration, intrusion detection)

- Process

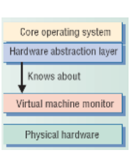


- Provides API interface
- Easier installation
- Leverage OS services (e.g., device drivers)
- Execution overhead (possibly mitigated by just-in-time compilation)

System-level Design Approaches

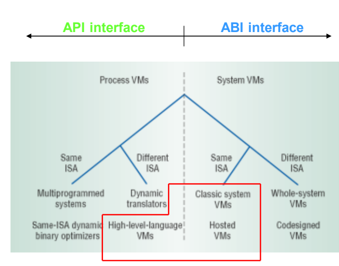


- Full virtualization (direct execution)
  - Exact hardware exposed to OS
  - Efficient execution
  - OS runs unchanged
  - Requires a "virtualizable" architecture
  - Example: VMWare



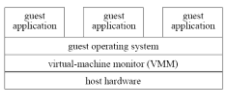
- Paravirtualization
  - OS modified to execute under VMM
  - Requires porting OS code
  - Execution overhead
  - Necessary for some (popular) architectures (e.g., x86)
  - Examples: Xen, Denali

Design Space (level vs. ISA)

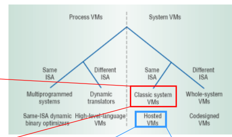


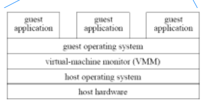
- Variety of techniques and approaches available
- Critical technology space highlighted

System VMMs



Type 1



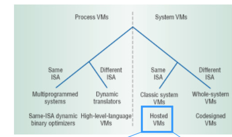


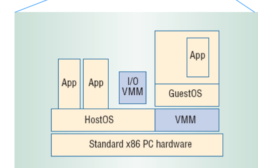
Type 2

- Structure
  - Type 1: runs directly on host hardware
  - Type 2: runs on HostOS
- Primary goals
  - Type 1: High performance
  - Type 2: Ease of construction/installation/acceptability
- Examples
  - Type 1: VMWare ESX Server, Xen, OS/370
  - Type 2: User-mode Linux

Hosted VMMs

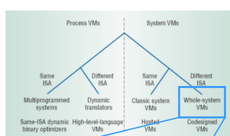
- Structure
  - Hybrid between Type1 and Type2
  - Core VMM executes directly on hardware
  - I/O services provided by code running on HostOS
- Goals
  - Improve performance overall
  - leverages I/O device support on the HostOS
- Disadvantages
  - Incurs overhead on I/O operations
  - Lacks performance isolation and performance guarantees
- Example: VMWare (Workstation)

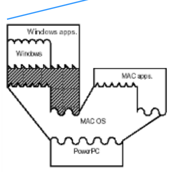




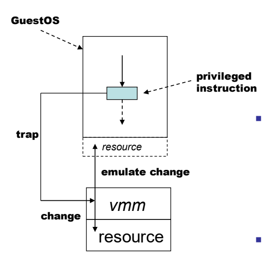
Whole-system VMMs

- Challenge: GuestOS ISA differs from HostOS ISA
- Requires full emulation of GuestOS and its applications
- Example: VirtualPC



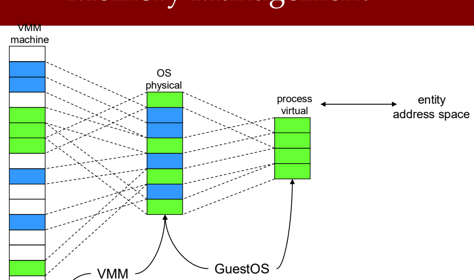


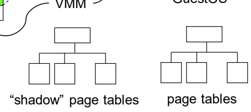
Strategies



- De-privileging
  - VMM emulates the effect on system/hardware resources of privileged instructions whose execution traps into the VMM
  - aka trap-and-emulate
  - Typically achieved by running GuestOS at a lower hardware priority level than the VMM
  - Problematic on some architectures where privileged instructions do not trap when executed at deprivileged priority
- Primary/shadow structures
  - VMM maintains "shadow" copies of critical structures whose "primary" versions are manipulated by the GuestOS
  - e.g., page tables
  - Primary copies needed to insure correct environment visible to GuestOS
- Memory traces
  - Controlling access to memory so that the shadow and primary structure remain coherent
  - Common strategy: write-protect primary copies so that update operations cause page faults which can be caught, interpreted, and emulated.

Memory Management





- Isolation/protection of Guest OS address spaces
- Efficient MM address translation